東海大學資訊工程學系研究所

碩士論文

指導教授：林祝興 博士

Dr. Chu-Hsing Lin

圖形處理器在 RSA 加密演算法應用之研究

RSA Encryption Algorithm Based on Graphic Processing Units

研究生：蔡奇軒

中 華 民 國 一 ◯ 二 年 六 月

中文摘要

在資訊發達科技進步的現代，人們日益仰賴電腦。由於資訊的流通量越來越龐大,資訊安全已成為重要的議題;也由於科技進步，竊取資料的技術也越來越先進，重要的個資以及機密文件若無妥善保管，就容易遭受有心人士取得並做不當的利用。為了有效的保護重要資料，將資料做加密運算是現今資訊界最重要的一環，而加密運算的效能與安全性也是資訊安全領域上重要的課題之一。

而在本文中我們藉由圖形處理器(Graphic Processing Unit, GPU)配合著中國餘數定理(Chinese Remainder Theorem, CRT)與蒙哥馬利運算(Montgomery Reduction)加速之 RSA 演算法，與以一般的 CPU 做運算之 RSA 演算法的效能比較。並且藉由執行運算實驗所取得的效能數據，驗證這幾種演算法的效能高低。

而從這些實驗中我們得到了一些研究結果，在結合了蒙哥馬利與中國餘數定理上之 RSA 演算法，與 GPU 相容性最佳。其效能將會比單純使用蒙哥馬利運算或單使用中國餘數定理還來的高。所以使用 GPU 平行 RSA 演算法，將 RSA 改寫成蒙哥馬利運算與中國餘數定理，會得到最大效能的產量。

關鍵詞：RSA 加密、CUDA、中國餘數定理、蒙哥馬利運算

ABSTRACT

Along with the development of modern information technologies, people more and more rely on computers. Due to the ever-growing circulation of information, security problems have become important issues. If important personal information and confidential documents have not been taken good care, they might be accessed by unlawful persons and used improperly. As the major role in providing effective protection of information, encryption algorithm becomes the most important part of the IT industry.

In this thesis, we used Graphic Processing Unit with the Chinese Remainder Theorem, and Montgomery reduction acceleration for the RSA algorithm. Furthermore, a general CPU was also used to compute RSA algorithms. We compared the performance of RSA algorithms in various environments, for which the data generated from the implementation to confirm the efficiencies among these different algorithms.

We got some results from the experiments, in conjunction with the Montgomery and the Chinese Remainder Theorem on the RSA algorithm, the GPU compatibility of the best. Its performance may be higher than using the Chinese Remainder Theorem or the Montgomery

algorithm only. So based on the GPU parallel architecture, by using cooperatively the Montgomery algorithm and the Chinese Remainder Theorem, RSA algorithm will get the maximum performance yield.
Keywords: RSA, CUDA, Chinese Remainder Theorem, Montgomery reduction

致謝

兩年的碩士生涯即將結束，求學階段也告一段落。首先在此最先要感謝的是我的家人，在於經濟與心靈上提供了無以倫比的支援。阿嬤我準備畢業了，謝謝您的鼓勵與支持，我會更加成長茁壯。這兩年碩士生涯，最感謝的即是我的授業恩師—林祝興老師，老師教導不只有課業上的知識，還有時常與我們分享有關心靈的成長與做人處世的態度，使我受益良多。再來是感謝賴威伸老師以及劉榮春老師，在於我論文的撰寫上提供了明確的方向以及良好的建議。還有口試當天撥冗前來參加之口試委員：詹進科教授、楊中皇教授、賴威伸教授、劉榮春教授，以及鎮宇學長的幫忙與指導，使此篇論文更加完備與充足。

再來是感謝實驗室裡一起奮鬥的好夥伴們，成熟穩重愛喝老人茶的明竑哥、熱心助人練功很勤勞的侑廷、脾氣好到被叫蛤蜊臉的俊良、只有帥可以說嘴的紘侑、興趣相似以及繪圖神手的傑立、資訊領域無敵的文琛。還有已經畢業的學長姐們：講屁話嗆人兼鼓勵的泓彥、雖然肥胖但有時候很貼心的棠濰哥、在我家附近工作的信斌以及留在學校當助教很強大的學姊。謝謝你們這些年來的包容、鼓勵以及幫忙，人很好耶!最後再次感謝父母、師長及親友們，在此致上最高的感謝。

蔡奇軒於東海大學研究所　　102 年 7 月

目錄

圖目錄

表目錄

第一章 簡介

在現今台灣科技產業發展快速，在此環境下行動裝置-像手機或者 PDA 等，以及網際網路的發達，資訊流通量與以前時代相比高出不少，資訊流通速度也逐年加快。正因於資訊流通速度大增，如果重要資料或者機密文件沒有做好保護，很容易遭受有心人士竊取或者惡意壞資料，間接造成使用者的權益受損。

也正因如此資訊安全的重要性也漸漸浮現出來，在法律上制定了個人資料保護法來保護個資的安全，而在科技方面則有各種不同的加密演算法用以加密資料，增強資料的安全性。在加密演算法上最主要效能取決於演算法的複雜度與否以及執行時間的快慢，演算法複雜度越高則文件的安全度也越發安全，而執行時間越短則能在相同時間之下處理越多資料，間接提升加密效率。但是越複雜之演算法其執行時間也相對變長，所以追求高安全性並且高效能之演算法一直是資訊安全領域所探討的重要議題。

在本論文上，我們探討了 RSA 演算法在不同環境下分別取得執行效能之數據，並且進行評估與分析。而執行環境分別是一般 CPU 運算環境以及配合 GPU 平行運算環境，在 CPU 運算環境中我們將執行原本的 RSA 演算法，並無使用 GPU 做輔助運算而只採用 CPU 作執行之動作。而在 GPU 平行運算環境中，我們先將 RSA 演

算法改寫成適合平行運算之演算法，再將需要大量運算的部分移入 GPU 中做平行運算，最後從兩種環境中取得效能比較與時間分析。

在本論文中第二章會介紹相關背景知識，第三章則是介紹研究方法以及環境配置，第四章是實驗結果以及分析，最後則是結論與未來展望。

第二章 相關文獻

在這一章節講到本論文所參考到之相關資料與技術，在 2.1 節會介紹基本之 RSA 演算法，以及在傳輸資料時將如何執行 RSA 之運算流程。2.2 節會介紹到 GPU 之硬體架構，以及執行 CUDA 程式時之執行流程與程式架構。2.3 節則是說明何謂平行運算以及介紹各種不同的運算架構，並且在此說明 GPU 在執行程式時比較傾向於何種運算架構，以及在此章介紹平方與乘法演算法，並且說明如何將此演算法應用在 RSA 演算法上。2.4 節介紹到蒙哥馬利運算，此運算在硬體應用上可以有效加速執行效率，而在此章提到蒙哥馬利乘法運算以及演算法。最後在 2.5 節提到了中國餘數定理，在 RSA 演算法中我們也會使用到此定理去減低 bit 數以提高效能。

2.1 RSA 演算法[2][22][23][24]

RSA 是由 Ron Rivest、Adi Shamir 與 Leonard Adleman 在 1978 年所研發推展。由於此演算法最大的特點在於加密金鑰與解密金鑰並非同一把，與一般對稱性加密演算法差異很大，因而又被稱之為非對稱性金鑰加密演算法，或者是公開金鑰加密演算法。

RSA 在現今社會上的應用廣泛，像是電子商務、網際網路安全協定 SSL 與數位簽章等，皆可看見 RSA 的應用。RSA 的安全性建

立在兩個大質數的相乘上，由於要將一大整數做因數運算拆成兩整數具有一定的困難度，安全性也建立在此之上。而 RSA 的執行步驟分成以下三種：金鑰生成、公鑰加密與私鑰解密。當接收端準備從發送端接收訊息時，發送端會將接收端所產生的公鑰去將訊息做加密，再將加密後之訊息傳遞給接收端。而接收端接收到加密訊息時，則透過自己的私鑰做解密，即可得到原始的訊息。而詳細流程則記錄於表 2-1 中。

RSA 金鑰產生
隨機選擇兩質數 p 與 q，而且 p 不等於 q。
計算$N = p * q$，並使用 N 來運算出公鑰與私鑰。
計算$\varphi(N) = (p-1)(q-1)$，φ代表尤拉函數。
選擇一整數 e，使得$1 < e < \varphi(N)$且$GCD(e, \varphi(N)) = 1$，此時(e,N)即為公鑰。
計算 d，$d \equiv e^{-1} \bmod \varphi(N)$，即取得私鑰(d,N)。
加密方式
$C \equiv M^{e} \bmod N$
解密方式
$M \equiv C^{d} \bmod N$

表 2-1 RSA 金鑰產生與傳送流程

2.2 GPU[20][21][27]

隨著現代科技進步，也有越多的事件需要精密且大量的運算，例如生物科技、天氣預報、太空科技等皆需要做大量的運算才有辦法得知正確的結果並做其他用途。

而此時可以運用 GPU 去支援運算，由於 GPU 相對於 CPU 在處理能力以及記憶體運用上較具有優勢，GPU 在設計上藉由大量的邏輯運算單元去提高效能，與 CPU 使用較為複雜的 control 與 cache 來提高效能不同。GPU 可以藉由此種配置進行平行運算以及大量處理單一指令動作達到縮短時間並且提高效能。在圖 2-1 則是比較 CPU 與 GPU 的算術邏輯單元的數量，GPU 具有較多的算術邏輯單元，但是每一橫列只能執行單一指令。

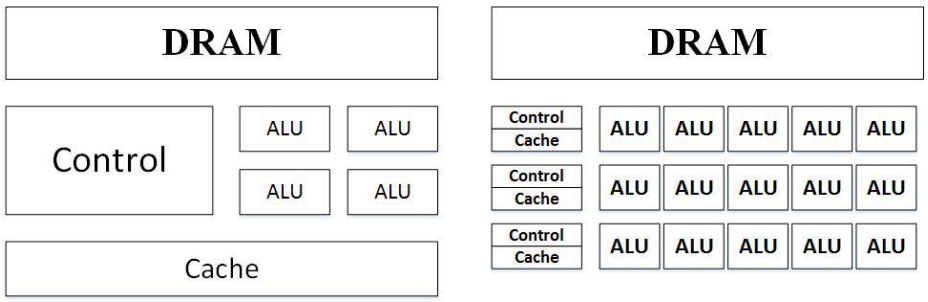

圖 2-1 CPU(左)與 GPU(右)的設備配置

2.2.1 CUDA[20][21][27]

CUDA(Computer Unified Device Architecture,統一計算架構)是 Nvidia 公司為了 GPU 所提出的一種架構,此架構語言比較偏向與 C 語言。Nvidia 藉由 CUDA 去開發適合的軟體與開發環境，圖 2-2 為 CUDA 架構的圖解。

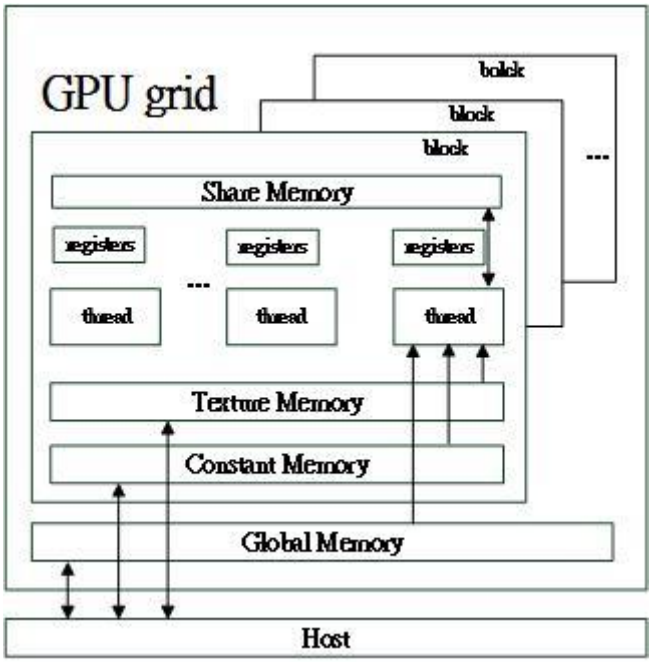

圖 2-2 CUDA 架構圖示

由此圖中可以知道 GPU 內具有多個 Multiprocessor，在每一個 Multiprocessor 內包含了四種的儲存設備，分別是 Share Memory、Texture Memory、Constant Memory 與 Register。而在每一個 block

中也建立了多個 thread。當 CUDA 在執行程式時，依照宣告的 grid 不同，如果在 GPU 程式執行時只宣告一個 grid，則此 grid 中的每個 block 中之每一個 thread 皆執行相同之指令，此時比較偏向於 SIMD。而當程式宣告多個 grid 時，不同的 grid 可以執行不同之指令，而 grid 中之的 thread 則是執行相同的指令，此時則是傾向於 MIMD 之架構。

而當程式在執行到 CUDA 程式時，CPU 端也就是 host 端會先將 GPU 端所需要的資料讀取入 GPU 中之 global memory，global memory 再將資料存入 block 中之 share memory。而每個 block 中之 thread 則抓取自己所需要之資料開始執行，並且將執行後結果存至各自 block 中之 share memory。等到整體程式執行結束，再將 share memory 中之結果存回 global memory，global memory 會將得到之結果回傳至 host 端上，並且繼續執行程式或者輸出結果。而資料除了讀取至 share memory 外，也可以將資料放置 global memory 直接由 thread 作抓取執行之動作。

在 2.2.2 與 2.2.3 CUDA 程式模型這兩章節中，我們使用一維陣列之加法與二維陣列之加法模型去解釋 CPU 程式設計方面與 GPU 程式設計方面之差異性。

2.2.2 CUDA 程式模型-一維陣列

此例子則是先令兩陣列 a[n]與 b[n]，使此兩陣列加總後之結果存至陣列 c[n]裡，表 2-2 與表 2-3 分別為一般程式模型以及 GPU 程式模型做比較。

表 2-2 CPU 一維陣列相加

```
int main(){
    for(int i=0;i<n;i++)
        c[i]=a[i]+b[i];}
```

表 2-3 CUDA 一維陣列相加

```
int main(){
    Kernel<<<1,N>>>(a,b,c);}


_global_ void Kernel(int *a, int *b, int *c){
    int i = threadIdx.x;
    c[i]=a[i]+b[i];}
```

一維陣列在一般程式上做加總時，只需要一個迴圈即可將全部陣列加總起來存至 c 陣列上。而 GPU 方面則需要先宣告 GPU 程式名稱，以及宣告會使用幾條 thread 去做執行。在上述例子中 Kernel 即是 GPU 端之程式，而<<<1,N>>>所代表之意思則是宣告此程式

會使用到多少 block 數以及多少 thread 數，而在上述例子中我們則宣告 1 個 block 數，並在在每一個 block 中存在著 N 條 thread。此時再將 GPU 端所需要之資料讀取進入 GPU 中執行。

將資料讀取進入 GPU 端程式後，會先將每條 thread 之 id 宣告存至變數 i 上，每條 thread 依照自己 id 去抓取所負責之資源並做運算，而此時每一條 thread 可以同步執行運算，即可達到平行化運算之要求與架構。圖 2-3 即是 GPU 模型執行一維陣列時之示意圖。

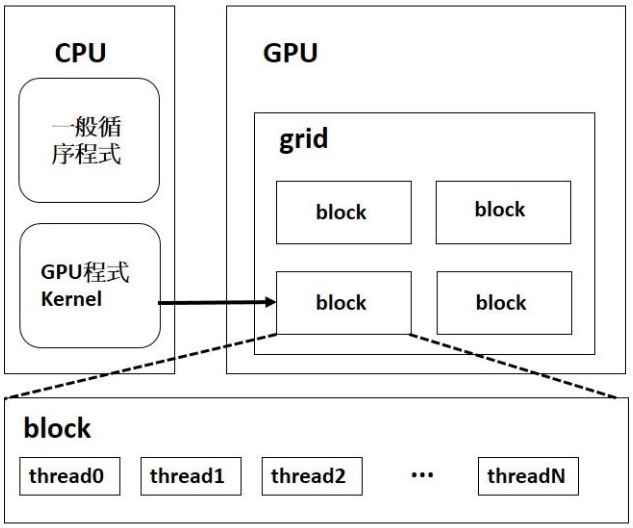

圖 2-3 GPU 流程圖一維陣列加法示意圖

2.2.3 CUDA 程式模型-二維陣列

此例子則是先令兩陣列 a[n][n]與 b[n][n]，使此兩陣列加總後之結果存至陣列 c[n][n]裡，表 2-4 與表 2-5 分別為一般程式模型以及 GPU 程式模型做比較。

表 2-4 CPU 二維陣列相加

```
int main(){
for(int i=0;i<n;i++)
    for(int j=0;j<n;j++)
        c[i][j]=a[i][j]+b[i][j];}
```

表 2-5 CUDA 二維陣列相加

```
int main(){
    dim3 dimBlock(N,N);
    Kernel<<<1,dimBlock>>>(a,b,c);}

_global_ void Kernel(int *a, int *b, int *c){
    int i = threadIdx.x;
    int j = threadIdx.y;
    c[i][j]=a[i][j]+b[i][j];}
```

二維陣列在一般程式與一維陣列不同之處在於，二維陣列使用兩個迴圈才能將全部作加總，而一維陣列則是使用一個迴圈即可執行完成。二維陣列時間複雜度為 $O(N^2)$而一維陣列時間複雜度為 $O(N)$，所以由此可知二維陣列其運算時間會比一維陣列運算時間來的更多。

而 GPU 方面則是先使用 dim3 去宣告，dim3 類型是基於 uint3 定義之向量類型，是相當於使用 3 個 unsigned int 型所組成之向量結構體。dim3 dimBlock(N,N)意思則是說宣告一個名稱為 dimBlock 之向量，並且大小為N × N。所以在此程式則是宣告一個 block，此 block 內含N × N條 thread，宣告完以後則是將 GPU 所需要之資料讀取進入 GPU 中。

將資料讀取進入 GPU 端程式後，與一維陣列不同，此時會將每條 thread 之 id 宣告存至變數 i 上以及變數 j 上，每條 thread 依照自己 id 去抓取所負責之資源並做運算，而此時每一條 thread 可以同步執行運算，即可達到平行化運算之要求與架構。與一般 CPU 所執行二維陣列加法比起來，運算速度加快許多。圖 2-4 即是 GPU 模型執行二維陣列時之示意圖。

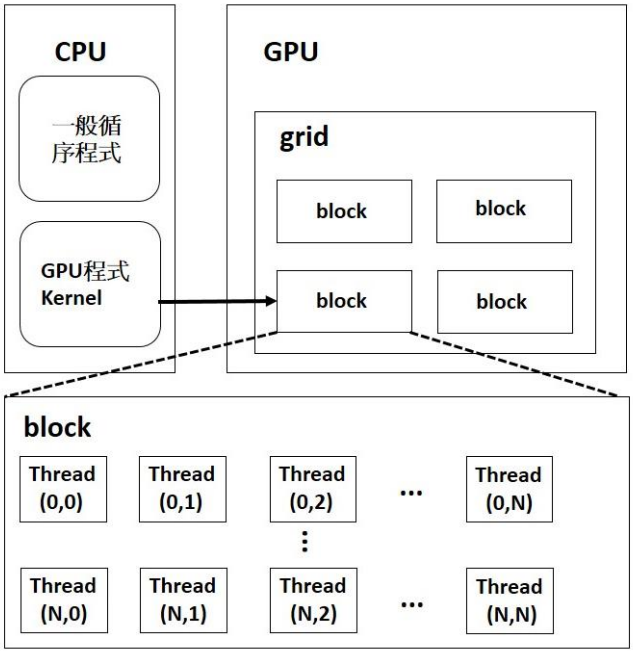

圖 2-4 GPU 流程圖二維陣列加法示意圖

2.3 平行運算[16][17][18][19]

在平行運算還沒發展之時，一般電腦僅僅依靠一顆 CPU 去處理應用程式。而程式的指令則是以循序讀取，在一個時間單位內只能處理一條指令並且執行，這時如果 CPU 效能不佳，處理此程式則需耗費大量時間。而平行運算發展後，我們可以將一個大程式拆解成小程式，而平行運算的優點在於可以同時讀取各個程式之指令，在同個時間單位內可以平行處理不同程式之指令並且執行，增加了效能並且也減少了運算時間。而圖 2-5 分別表示單一 CPU 運算以及多顆 CPU 執行平行運算之示意圖。

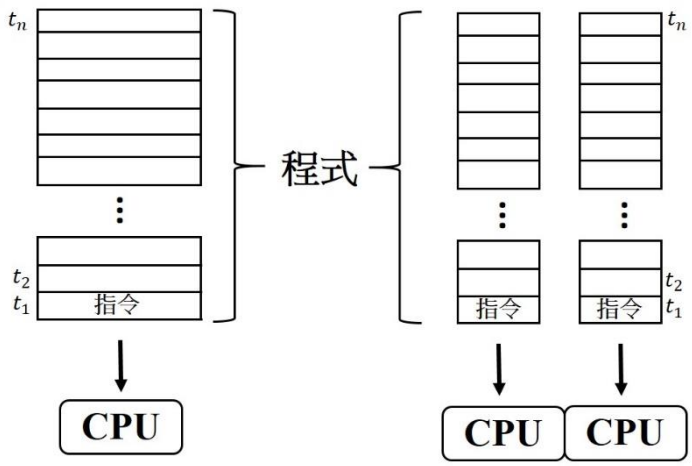

圖 2-5 左為單一 CPU 處理程式，右為多 CPU 處理程式

2.3.1 運算種類

根據 Flynn 所提出的分類，依照指令與資料的多寡，可將運算種類分成四種，我們則以下表來表示此四種不同之運算。並且以不同的圖示表示四種運算，而 CUDA 運算則傾向於 SIMD。

表 2-6 Flynn 提出之運算種類

名稱	意義
SISD	單一指令，單一資料，非平行運算
MISD	多重指令，單一資料，平行運算
SIMD	單一指令，多重資料，平行運算
MIMD	多重指令，多重資料，平行運算

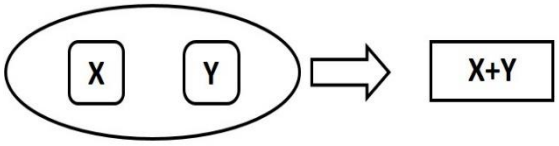

圖 2-6 SISD 運算示意圖，將單一資料 X 與 Y 執行 X+Y 之動作

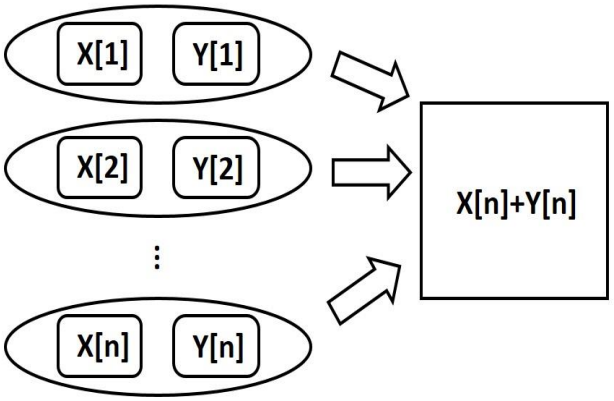

圖 2-7 SIMD 運算示意圖，將矩陣 X[n]內之值與 Y[n]內之值分別做加總

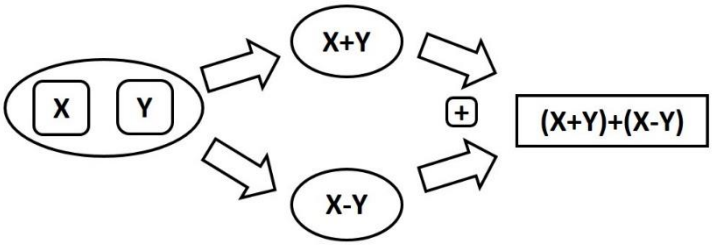

圖 2-8 MISD 運算示意圖，將單一資料 X 與單一資料 Y 個別做加法與減法，再將兩資料加總

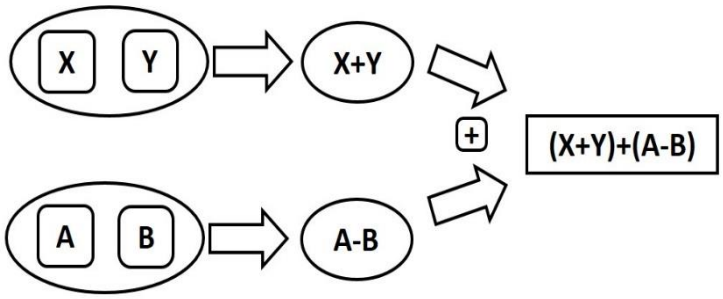

圖 2-9 MIMD 運算示意圖，將不同的資料 X、Y、A、B 分別運算，再將運算結果加總

2.3.2 平方與乘法計算[25][26]

RSA 演算法之運算可先使用平方與乘法計算(square and multiply computation)將之改寫，再將演算法裡有使用到指數運算與模數運算之部分改用蒙哥馬利運算來加速。圖 2-10 與表 2-7 分別是基本之平方與乘法計算之流程圖與演算法。下一節則將介紹使用蒙哥馬利運算加強改寫之 RSA 非對稱性演算法。

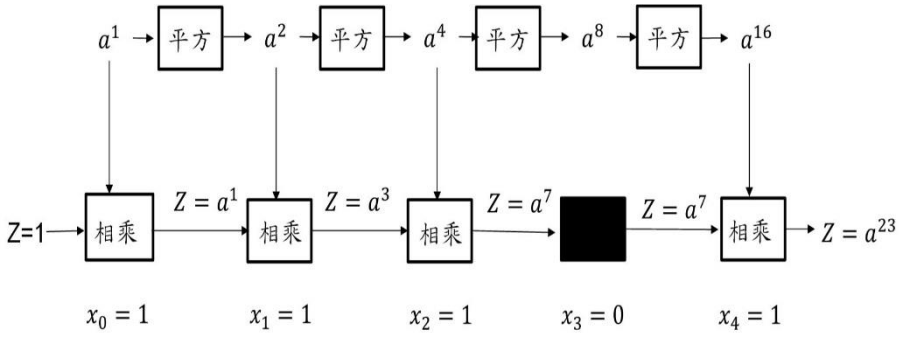

圖 2-10 平方與乘法計算流程圖

表 2-7 平方與乘法計算

b[0] = a;
total=1;
for i = 0 to k − 1 loop;
b[i + 1] = b[i] × b[i]
if $x_k = 1$;
total = x_k × b[i] × total;

在平方與乘法計算中，我們欲求出 Z=a^{x}，k 為 x 化作 2 進位之 bit 數，而流程圖則是以 x=23 為例執行此計算。而表 2-8 則是表示 RSA 演算法藉由平方與乘法計算改寫後之結果。我們處理$C = M^{e}(mod\ N)$，M 為明文，e 為公開金鑰，k 是公開金鑰 e 轉成二進制之 bit 數，N 則是 p 與 q 兩質數互乘之結果。

表 2-8 以平方與乘法計算改寫之 RSA 演算法

b[0] = M;
total=1;
for $i = 0$ to $k - 1$ loop;
$b[i + 1] = b[i] \times b[i](\text{mod N})$
if $e_k = 1$;
$\text{total} = b[i] \times \text{total}(\text{mod N})$;

表 2-9 以平方與乘法計算改寫之 RSA 演算法步驟

步驟一：CPU 設定陣列 b[i]=M ， $0 \le i \le k$ ，設定一 total=1。
步驟二：for$(0 \le i \le k)$//每一個 processor，平行化。
$b[i] \leftarrow M^{2^{i}}(mod\ N)$。
步驟三：CPU 判斷二進位指數，如果e_k=1，則執行$total$=$b[i]$×$total(\ mod\ N)$。

2.4 蒙哥馬利運算[6][8][9][10][11]

蒙哥馬利運算(Montgomery reduction)對於模數運算而言，是一種可以有效提升效能的演算法。在使用蒙哥馬利運算時，會使用一連串的加法運算與乘法運算取代比較複雜之除法運算。而蒙哥馬利運算也由於其運算特性，能使用硬體設備實現並且有效提升運算之效能。

而蒙哥馬利運算中最重要的運算則是蒙哥馬利乘法(Montgomery multiplication)，我們欲運算$c = a \times b$，則使用蒙哥馬利運算表示成$C = A \times B$。A 如欲使用 a 表示，則會給定一整數$a < n$，n 為 k-bit 的數，使用蒙哥馬利運算可算出：

$$A = a \times 2^k (\mathrm{mod}\ n)$$

B 也依此步驟求得：B 如欲使用 b 表示，則會給定一整數$b < n$，n 為 k-bit 的數，使用蒙哥馬利運算可算出：

$$B = b \times 2^k (\mathrm{mod}\ n)$$

由蒙哥馬利運算得到 A 與 B 後，我們即可求得$c = a \times b$，而使用蒙哥馬利運算後求得之 C 結果則是如下定義：

$$C = A \times B \times 2^{-k} (\text{mod } n)$$

將以上定義之結果運算後，即可得到$c = C \times 2^{-k} (\text{mod } n)$之結果。而以上所做之乘法，即是蒙哥馬利運算，圖 2-11 為蒙哥馬利運算示意圖。

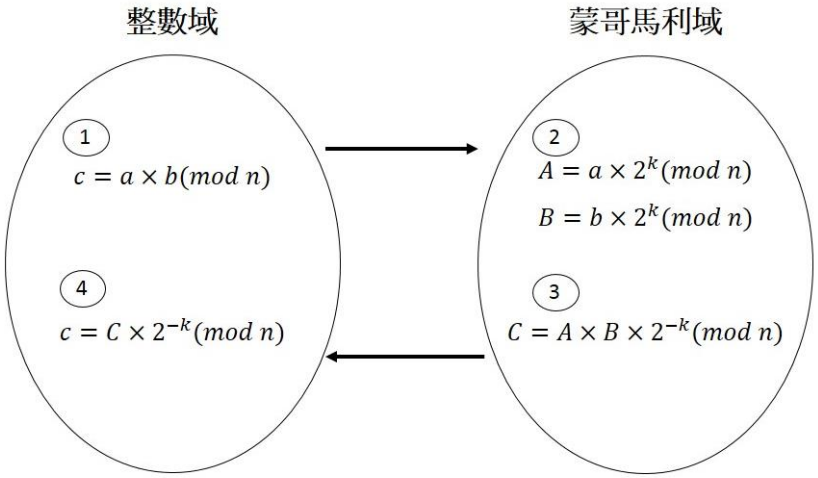

圖 2-11 蒙哥馬利乘法示意圖

由此示意圖可以得知，要在整數域執行$c = a \times b (\text{mod } n)$，此時可以將 a 與 b 投影至蒙哥馬利域上，經過運算得到 A 與 B。再將 A 與 B 作運算，可以得到 C。最後，再將 C 投影回整數域上

即可得到c。表 2-10 為其演算法所需用到之參數名稱與意義；表 2-11 則是蒙哥馬利乘法之演算法。

表 2-10 蒙哥馬利運算參數說明

參數名稱	參數意義
$c[i]$	矩陣 c 第 i 格
a_i	整數域代數 a 第 i-bit
b	整數域代數 b
n	模數 n
k	模數 n 之 bit 數

表 2-11 蒙哥馬利運算之乘法

Montgomery Multiplication(a,b,n)
c[0] = 0;
for i = 0 to k − 1 loop
p = (c[i] + $a_i \times b$)(mod 2);
c[i + 1] = (c[i] + $a_i \times b$ + p × n)(div 2);
end loop
return c[k];

蒙哥馬利運算只需要將整數域所需之整數輸入進去，即可得到 $c = a \times b \pmod{n}$之結果。蒙哥馬利運算藉由乘法與加法替代了除

法與指數運算，並且藉由左移與右移去實現 2 的 k 次方之運算。而此運算之所需要模數條件 n 必須是奇數，而 RSA 非對稱性演算法可以滿足此條件。這是因為 RSA 演算法需要大量的模數運算以及指數運算，而此時使用蒙哥馬利運算能與 RSA 演算法有效配合，並且提升 RSA 演算法的效能。

2.5 中國餘數定理[3][13][14][15]

中國餘數定理在西元 100 年左右由中國數學家所提出，由於是在中國數學著作「孫子算經」發現，所以又被稱作孫子餘數定理。其他例如「秦王點兵」、「韓信點兵」等，皆是中國餘數定理衍生出之問題。

孫子算經中之問題「有物不知其數，三三數之剩二，五五數之剩三，七七數之剩二。問物幾何？」，以現今數學觀點來看即是給定一整數 X，X 除以 3 餘 2，X 除以 5 餘 3，X 除以 7 餘 2，求 X 為多少？而解答則是$23 + 105\mathrm{k}$，k 為任意非負整數。而用數學表示方法描述此中國餘數定理問題[1]，則可以由下列題目敘述所表示之。

有若干個正整數，其兩兩互質，則我們可以定義為：$\mathrm{m}_1, \mathrm{m}_2, \dots, \mathrm{m}_\mathrm{n}$，且$1 < i, j < n$，最大公因數$\gcd(\mathrm{m}_\mathrm{i}, \mathrm{m}_\mathrm{j}) = 1, \mathrm{i} \neq \mathrm{j}$。對任意整數：$\mathrm{b}_1, \mathrm{b}_2, \dots, \mathrm{b}_\mathrm{n}$使用聯立同餘方程式求解 X，令$\mathrm{m} = \mathrm{m}_1\mathrm{m}_2 \dots \mathrm{m}_\mathrm{n}$，則有一解答 X 與全任意整數同餘，可寫成如下函數：

$$\begin{cases} \mathrm{X} \equiv \mathrm{b}_1 (\mathrm{mod}\ \mathrm{m}_1) \\ \mathrm{X} \equiv \mathrm{b}_2 (\mathrm{mod}\ \mathrm{m}_2) \\ \quad\quad \vdots \\ \mathrm{X} \equiv \mathrm{b}_\mathrm{n} (\mathrm{mod}\ \mathrm{m}_\mathrm{n}) \end{cases}$$

在此同餘聯立方程式下將會求得同一解 X，$0 < X < n - 1$。

中國餘數定理可使用高斯演算法去求解並證明，先令$r_i = \frac{m}{m_i}$以及$s_i = r_i^{-1} \bmod m_i$，$i = 1,2,\dots,n$。其運算方程式如下：

$$X = \left(\sum_{i=1}^{n} b_i r_i s_i \right) \bmod m$$

$$= (b_1 r_1 s_1 + b_2 r_2 s_2 + \cdots + b_n r_n s_n) \bmod m$$

根據此方程式，即可算出滿足同餘方程式之唯一解 X，且$0 \leq X < m$。

第三章 研究方法

此一章節介紹本論文所使用的研究方法，以及執行運算時所使用到之演算法。3.1 節先介紹到本論文所使用之環境以及硬體使用與架構，在 3.1.1 節則是說明蒙哥馬利運算如何結合 RSA 演算法，並且將 RSA 演算法改寫以加速運算效率。3.1.2 則是介紹到中國餘數定理與 RSA 演算法的結合與應用，中國餘數定理在於減少 RSA 演算法運算中之 bit 數，藉由明文與公開金鑰之 bit 數減少以提升效能。3.1.3 則是介紹到本論文所使用之中國餘數定理與蒙哥馬利運算加速之 RSA 演算法，在此章節中將使用中國餘數定理將明文與公開金鑰之 bit 數減少後，需要指數運算時再將此運算交至蒙哥馬利運算上去加速執行。

3.2 則是介紹到 CPU 環境，在此先介紹到程式使用環境以及使用何種硬體。而在 CPU 環境中我們則是將一般之 RSA 演算法放置 CPU 環境中執行，並且計算其執行時間並且記錄。最後再與 GPU 所執行出之時間做比較。

3.1 GPU 環境[4][5][7][12]

在執行 GPU 實驗之環境方面，我們使用 Nvidia Tesla C2075 運算處理器，CPU 則是 Intel® Core™2 Quad Processor Q8200，作業系統方面則使用 CentOS 6.3，CUDA drivers 與 samples 版本為 CUDA5，CUDA toolkit 則是採用了 CUDA4.2。

而 RSA 加密演算法是以 Ron Rivest、Adi Shamir 與 Leonard Adleman 所發表之演算法，再分別使用中國餘數定理以及蒙哥馬利運算進行改寫，使之符合 GPU 平行運算之架構。我們會使用蒙哥馬利乘法運算$C = M^e \pmod n$，並且以圖三表示 GPU 之運算流程以及圖四表示程式的階層架構圖。

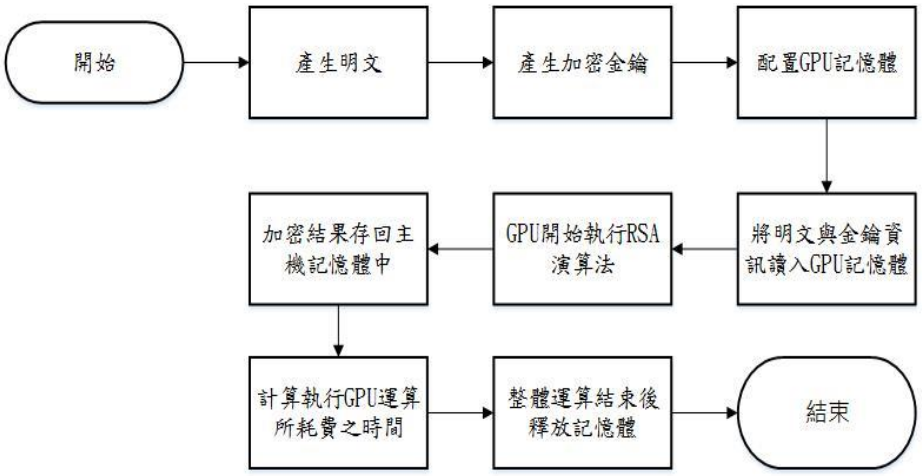

圖 3-1 GPU 運算流程圖

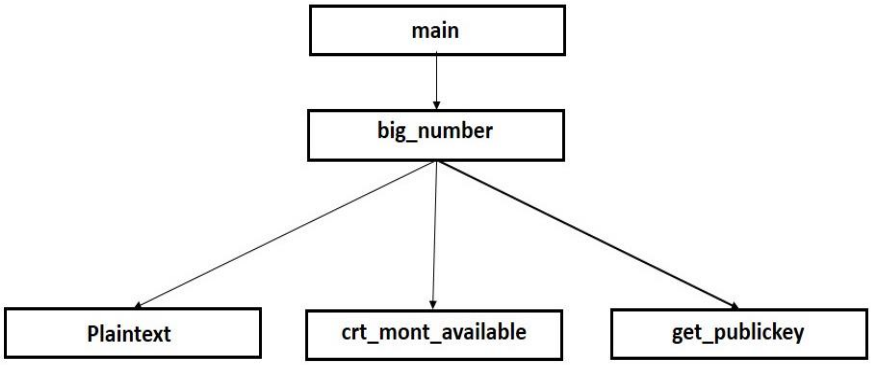

圖 3-2 程式階層架構圖

此主程式會先讀取大數運算之程式，並且將支援大數運算之程式讀取進來，使用 plaintext 產生 RSA 程式所需之明文，使用 get_publickey 產生加密所需之公開金鑰，以及最後驗證中國餘數定理結合蒙哥馬利演算法此程式可否執行，可執行即直接運算 RSA 加密程式並且計算執行時間。

在此會分別介紹使用蒙哥馬利運算改寫之 RSA 非對稱性演算法、使用中國餘數定理改寫之 RSA 非對稱性演算法，以及將上述兩者合成並改寫之 RSA 非對稱性演算法。

3.1.1 蒙哥馬利運算加速之 RSA[6][8]

在 RSA 演算法中 M 代表明文，C 代表密文，n 由兩個大質數 p、q 相乘可得，並將 n 設定成 k-bit 的模數。(e,n)表示接收端所產生之公開金鑰，而(d,n)則表示接收端所持有之私密金鑰。

而RSA的運算就是一個典型的模數運算與指數運算所結合之加密演算法，其運算方式與蒙哥馬利運算之條件相符合，故可以使用蒙哥馬利運算將RSA改寫，使用蒙哥馬利運算改寫加速之RSA演算法如下表3-2所示，並在表3-1介紹蒙哥馬利運算所需用到之參數名稱與說明。

表3-1 蒙哥馬利運算加速之RSA-參數說明

參數名稱	**參數意義**
montmul	執行蒙哥馬利乘法運算
k	指數 e 的 bit 長度
e[i]	指數 e 第 i 個 bit 之值

表3-2 蒙哥馬利運算加速之RSA

```
RSA_Montgomery(M,e,n){
K=2^{2k} (mod n);
X[0]=MontMul(K,M,n);
Y[0]=MontMul(K,1,n);
For i=0 to k−1 loop
    X[i+1]=MontMul(X[i],X[i],n);
    if e[i]=1 then
        Y[i+1]=MontMul(Y[i],X[i],n);
    if e[i]=0 then
        Y[i+1]=Y[i];
    end if;
end loop
C=MontMul(1,Y[k],n);
return C;}
```

而蒙哥馬利運算則是以平方與乘法計算為基礎，將 RSA 非對稱性演算法改寫。而藉由蒙哥馬利運算改寫後之 RSA 演算法，可以利用 GPU 去做平行運算而加快速度，此時可再配合中國餘數定理，加強 RSA 之平行度，可使運算效能再提升。

3.1.2 中國餘數定理加速之 RSA [15]

RSA 要配合中國餘數定理，還必須加上費馬小定理輔助運算才有辦法改寫成適合平行運算之架構，而費馬小定理與其推論之介紹如下表所示。

表 3-3 費馬小定理

選定一質數 p，存在任意整數 a 且 a 與 p 互質，則 a 滿足$a^{p-1} \equiv 1 \; mod \; p$。

根據費馬小定理所示，要導出 a 之反函數，只要將費馬小定理改寫成以下之函數$a^{p-2} \equiv a^{-1} \; mod \; p$，而可以$a^{p-2}$推算出$a^{-1}$。由以上定理可以導出以下推論。

如果任意整數 a 與 p 互質且$n \equiv m \; (\mathrm{mod}\; p-1)$，則可滿足$a^n \equiv a^m \; mod \; p$。

從以上推論可得知當模數為 p 時，其指數可以藉由模除以(p-1)使數值有效減少。在計算 RSA 演算法時可以根據此推論出之結果來做，以減少指數的長度並使之提高運算效能。另外，藉由以下推論，則可以將中國餘數定理應用在 RSA 演算法之上。

令密文為 C，則可以算出明文對$[C_p, C_q]$，$C_p = C\ (mod\ p)$且$C_q = C\ (mod\ q)$。而藉由以下的運算可以有效的減少指數運算 bit 數，而明文 M 可以在運算開始前先執行模數運算以減少 bit 數，能有效提高運算效率。

$$C_p = C\ (mod\ p) = \big(M^e\ (mod\ N)\big)(mod\ p) = M^e(mod\ p)$$

$$= M^{e\ (mod\ p-1)}(mod\ p) = M^{e_p}(mod\ p)$$

$$M_p = M\ (mod\ p)$$

$$C_p = {M_p}^{e_p}(mod\ p)$$

C_q也藉由以上之運算得知後，運用中國餘數定理則可算出 C。令$b_1 = C_p$，$b_2 = C_q$，$m_1 = p$，$m_2 = q$，$\mathrm{m} = m_1 \times m_2 = p \times q = N$，所以可以得到$r_1 = \frac{m}{m_1} = \frac{\mathrm{N}}{\mathrm{p}} = \mathrm{q}$，$r_2 = \frac{m}{m_2} = \frac{\mathrm{N}}{\mathrm{q}} = \mathrm{p}$，$s_1 = \mathrm{r}_1^{-1} \bmod \mathrm{m} = q^{-1}\ (\mathrm{mod\ p})$，$s_2 = \mathrm{r}_2^{-1} \bmod \mathrm{m} = p^{-1}\ (\mathrm{mod\ q})$則將以上算出後之數值全部帶入函數即可得：

$$\mathrm{C} = \left(\sum_{\mathrm{i}=1}^{2} \mathrm{b_i r_i s_I}\right) \bmod \mathrm{m} = (\mathrm{b_1 r_1 s_1 + b_2 r_2 s_2}\ ) \bmod \mathrm{m}$$

$$= \Big(\mathrm{C_p \times q \times q^{-1}\ (mod\ p)}\Big) + \Big(\mathrm{C_q \times p \times p^{-1}\ (mod\ q)}\Big)\ (\mathrm{mod\ N})$$

$$= \Big(\mathrm{C_p \times q \times q^{p-2}\ (mod\ p)}\Big) + \Big(\mathrm{C_q \times p \times p^{q-2}\ (mod\ q)}\Big)\ (\mathrm{mod\ N})$$

$$= \Big(\mathrm{C_p \times q^{p-1}\ (mod\ p)}\Big) + \Big(\mathrm{C_q \times p^{q-1}\ (mod\ N)}\Big)\ (\mathrm{mod\ N})$$

所以 RSA 經由中國餘數定理改寫後，由定理可知其運算$C = M^{e}(\mathrm{mod}\ n)$可拆解成兩個部分，表五即是中國餘數定理演算法。

表 3-4 中國餘數定理改寫之 RSA 演算法

crt_operator(M,e){

$M_p = M\ (\mathrm{mod}\ p)$;

$M_q = M\ (\mathrm{mod}\ q)$;

$e_p = e(\mathrm{mod}\ p - 1)$

$e_q = e(\mathrm{mod}\ q - 1)$

$C_p = {M_p}^{e_p}(\mathrm{mod}\ p)$;

$C_q = {M_q}^{e_q}(\mathrm{mod}\ q)$;

$C = \left[C_p \times (q^{p-1}\ \mathrm{mod}\ N) + C_q \times (p^{q-1}\ \mathrm{mod}\ N)\right] \mathrm{mod}\ N$;}

而e_p 與e_q 兩數皆比原本的 e 還小，且M_p與M_q兩數也皆比原本之 C 還來的小。也由於整體資料大小皆小於原本的 M 與 e，可使執行效能提升。

3.1.3 中國餘數定理與蒙哥馬利運算加速之 RSA

我們將前兩章介紹到之加速計算法做結合，以中國餘數定理所改寫之 RSA 非對稱性演算法為主，蒙哥馬利運算為輔。我們將中國餘數定理中計算M_p與M_q這兩部分的運算更改成蒙哥馬利運算，藉由 GPU 的平行運算能力提升改寫後之 RSA 運算效能。

將 RSA 以中國餘數定理改寫後，再配合著蒙哥馬利運算，此兩種技巧融合改寫之演算法如表 3-5 所表示。

表 3-5 以中國餘數定理與蒙哥馬利運算加速之 RSA

crt_mont(M,e){
$M_p = M \pmod p$;
$M_q = M \pmod q$;
$e_p = e \pmod{p-1}$
$e_q = e \pmod{q-1}$
C_p = RSA_Montgomery (M_p,e_p,p);
C_q = RSA_Montgomery (M_q,e_q,q);
$C = \left[C_p \times (q^{p-1} \bmod N) + C_q \times (p^{q-1} \bmod N)\right] \bmod N$;}

我們實驗中所測試之演算法為第三種中國餘數定理與蒙哥馬利運算改寫之 RSA 非對稱性演算法。測試環境中設定之金鑰長度為 1024-bit 與 2048-bit，明文訊息長度則是以每 32-bit 為一單位從 32-bit 測試至 1024-bit，而實驗結果則是將 RSA 程式執行之時間與每秒能執行多少訊息記錄起來並整合成折線圖。

3.2 CPU 環境

在 CPU 環境中，裡面實驗所使用之 CPU 為 Intel® Core™2 Quad Processor Q8200。而 RSA 程式方面，我們將 RSA 非對稱性演算法放置於 CPU 執行。

在 CPU 執行 RSA 非對稱性演算法，此演算法將依照 Ron Rivest、Adi Shamir 與 Leonard Adleman 對於 RSA 非對稱性演算法之研究發展為基礎寫成。測試環境中設定之金鑰長度為 1024-bit 與 2048-bit，明文訊息長度則以每 32-bit 為一單位從 32-bit 測試至 1024-bit，並且將個明文訊息長度所執行之效能逐步紀錄下來，並製作成折線圖以利分析與探討。

第四章 研究結果

我們實驗設定金鑰長度 1024-bit 與 2048-bit 兩種，並且將 GPU 與 CPU 的執行時間與執行效率紀錄下來，執行效率我們則是使用訊息長度除以執行時間做記錄。而明文的長度以每 32-bit 為一單位從 32-bit 測試至 1024-bit。在執行程式方面，我們分別使用 CPU 執行 Ron Rivest、Adi Shamir 與 Leonard Adleman 在 1978 年所研發推展架構之 RSA 非對稱性演算法，以及使用 GPU 配合平行運算之中國餘數定理與蒙哥馬利演算法改寫之 RSA 演算法此二種。圖表參數名稱與意義記錄在下表中。執行時間分別記錄至圖 4-1 與圖 4-2，執行效率則是紀錄至圖 4-3 與圖 4-4。

在執行時間方面，圖 4-1 顯示當金鑰長度為 1024-bit 時，GPU 執行時間則會比使用 CPU 執行時間還來的減少許多。而圖 4-2 則是顯示當金鑰長度為 2048-bit 時，GPU 執行時間則會比使用 CPU 執行時間還少，效能也顯著提升。

而在執行效率運算上，CPU 運算無論訊息量大或者訊息量小，其運算效能皆是處在相同的狀態，這代表 CPU 處理小量運算與大量運算時速率皆相同。而 GPU 當訊息量為小時效能與 CPU 之效能差異不大，但在於處理小訊息時其效能成長會比較快。當訊息量大時，GPU 處理效率比 CPU 來的要高，但是由於訊息量趨於更大，所以處理效率漸於趨緩。

在做明文長度較短的運算時，由於明文與金鑰將由主機傳送至 GPU 中做運算，再將運算後之結果回傳至主機記憶體。大部分的運算時間耗損在傳輸資料上，故資料量較小時，直接在 CPU 上做運

算會比傳輸至 GPU 做運算效能要來的高。但是當運算資料量大，此時大量的時間皆消耗在執行運算上，傳輸時間反而可以忽略不計，這時傳輸至 GPU 作平行運算，其執行效能會比直接使用 CPU 作運算要來得高。

表 4-1 圖表意義說明

圖表名稱	圖表意義
RSA on CPU	以一般 CPU 執行 RSA 結果做基準
CRT_mont on CPU	GPUc2075 執行蒙哥馬利運算結合中國餘數定理加速

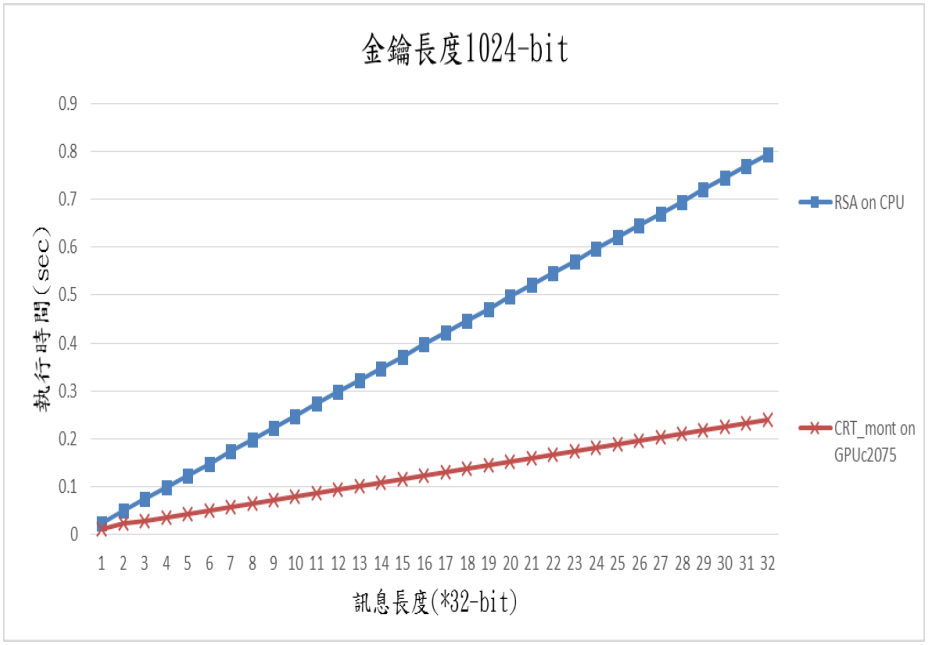

圖 4-1 金鑰長度 1024-bit 執行時間

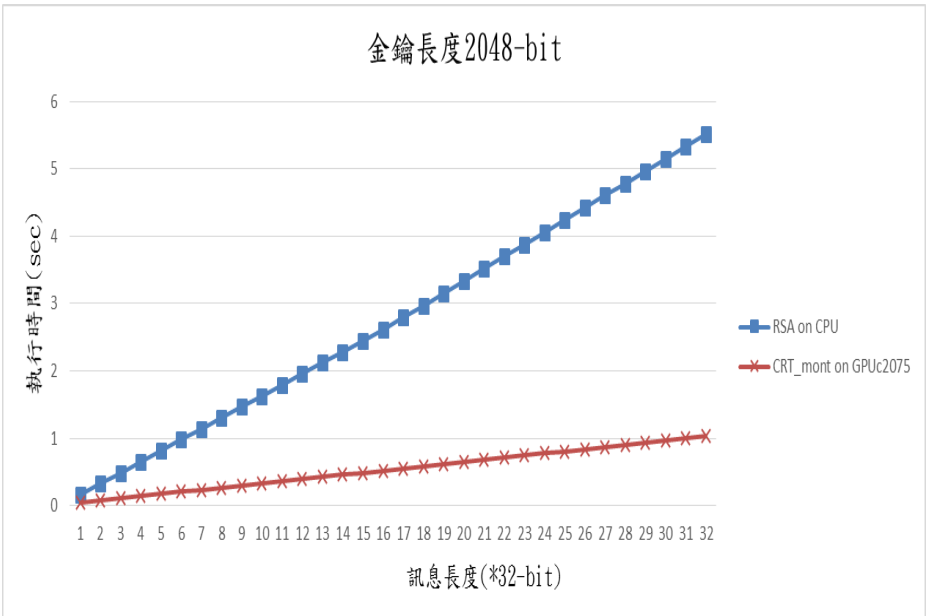

圖 4-2 金鑰長度 2048-bit 執行時間

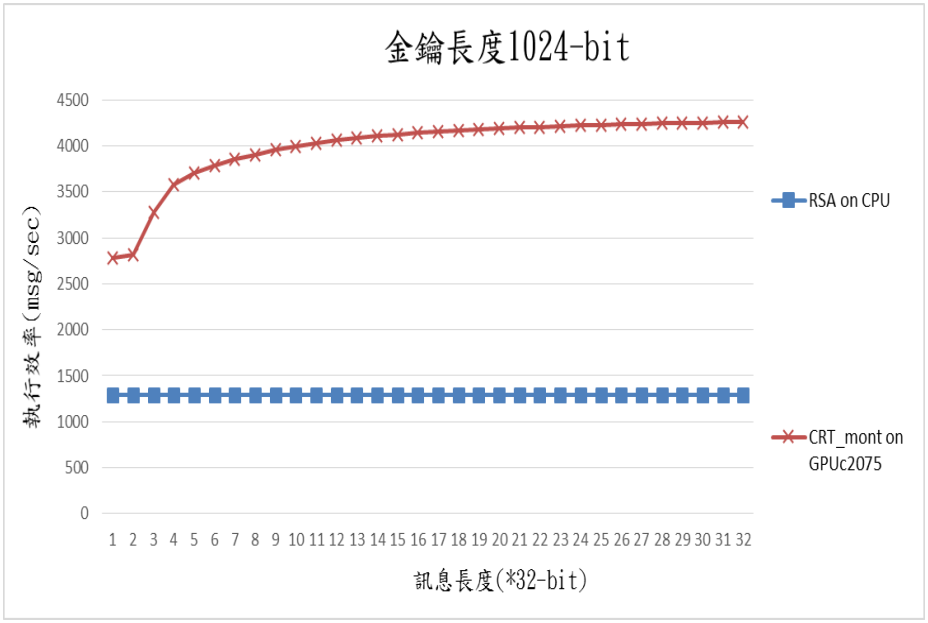

圖 4-3 金鑰長度 1024-bit 執行效率

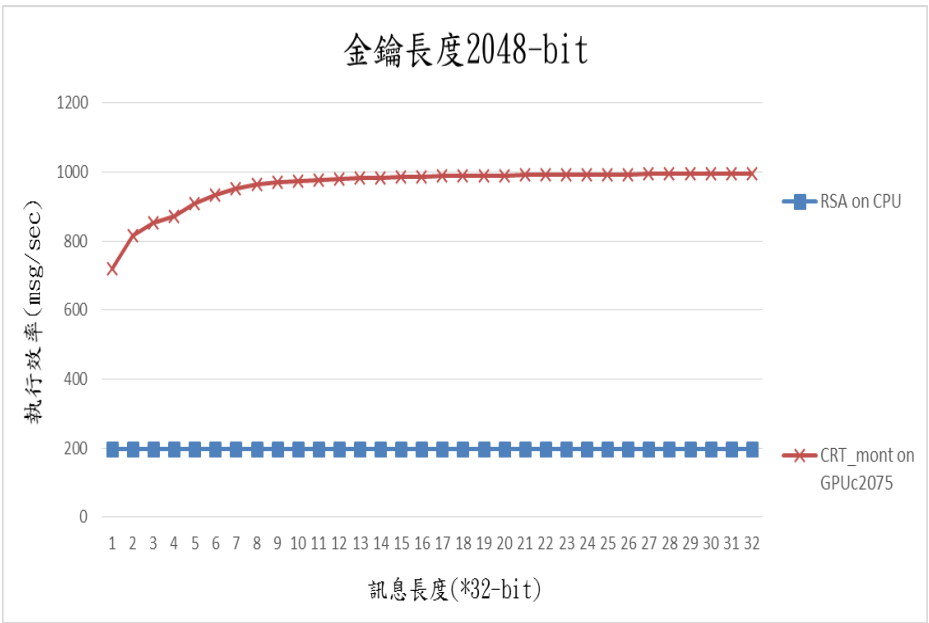

圖 4-4 金鑰長度 2048-bit 執行效率

第五章 結論與未來展望

本論文的重點在測試不同的環境使用 RSA 演算法之效率，其中環境包含了一般 CPU 運算環境以及 GPU 執行運算環境這兩種，並且將不同環境中的執行效能與結果做比較並提出結論。從我們的研究結果可以得知，金鑰長度是影響加速快慢最主要的條件，而訊息長度則是影響快慢的第二條件。從實驗結果中可以得知，在資料量較少時，直接採取 CPU 做 RSA 運算將會比使用 GPU 作平行運算要來得快。而當資料量較為龐大時，使用 GPU 做平行運算將會比使用 CPU 直接做運算效率要來得高。而現今科技發達，傳輸至網路上之傳輸資料量越來越大，所以在需要處理大量的訊息時，選擇 GPU 做加密運算將會優於僅使用 CPU 做加密運算的方式。

在本篇論文中我們將 RSA 非對稱式演算法改寫，藉由蒙哥馬利運算以及中國餘數定理去改寫 RSA 演算法，以實現平行運算之構想。而改寫後之 RSA 演算法也如預期般執行效能變得更佳，執行時間較一般 CPU 執行之 RSA 演算法更有效率。在研究結果中可以發現當金鑰長度較長，或者加密之資料檔案較為龐大，使用 GPU 並且配合著改寫後之RSA 演算法執行平行運算，其執行效能與 CPU 比較起來，具有更顯著之效能提升。

未來我們將依照此研究結果繼續發展，接著嘗試結合伺服器輔助運算與 GPU 運算。並且配合著雲端儲存系統，將大量的運算傳送至雲端伺服器作運算，並且配合 GPU 加速平行化處理。運算後之結果可以先儲存至雲端伺服器上，當使用者需要時再從雲端伺服器下載來即可使用執行。此種作法不必將資料儲存在手機上，減少

手機上的儲存量並且提升運算效能。此想法將可應用於小型設備或手持裝置上，即使設備本身受限於運算量不足的問題，也能藉此構想而使小型設備與手持裝置具有高度的安全性，由此構思可預見其效能將會比單單使用 GPU 或者伺服器輔助運算要來得更好。

參考文獻

[1] Sung-Ming Yen, Seung-Joo Kim, Seon-Gan Lim, and Sang-Jae Moon, "RSA speedup with Chinese remainder theorem immune against hardware fault cryptanalysis," IEEE Trasaction on Computers, Vol. 52, No. 4, pp. 461-472, April, 2003.

[2] Ron Rivest, Adi Shamir, and Leonard Adleman, “A method for obtaining digital signature and public key cryptosystems,” Communications of the ACM, Vol. 21, No. 2, pp. 120-126, Feb, 1978.

[3] Chung-Hsien Wu, Jin-Hua Hong, and Cheng-Wen Wu, "RSA cryptosystem design based on the Chinese remainder theorem," Proceedings of the 2001 ACM Asia and South Pacific Design Automation Conference, pp. 391-395, Feb, 2001.

[4] Wen-Jun Fan, Xu-Dong Chen, and Xue-Feng Li, "Parallelization of RSA algorithm based on compute unified device architecture," 9th IEEE International Conference on Grid and Cooperative Computing (GCC), pp. 174-178, Nov, 2010.

[5] Keon Jang, Sang-Jin Han, Seung-Yeop Han, Sue Moon, and Kyoung-Soo Park, "SSL shader: cheap SSL acceleration with commodity processors," Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, pp. 1-13, 2011.

[6] C. McIvor, M. McLoone and J.V. McCanny, "Modified Montgomery modular multiplication and RSA exponentiation techniques," IEE Proceedings- Computers and Digital Techniques, Vol. 151, No. 6, pp. 402-408, Nov, 2004.

[7] Chin-Chen Chang, Ying-Tse Kuo and Chu-Hsing Lin, "Fast algorithms for common-multiplicand multiplication and exponentiation by performing complements," 17th IEEE International Conference on Advanced Information Networking and Applications, pp. 807-811, Mar, 2003.

[8] Daniel Page and Nigel P. Smart, "Parallel cryptographic arithmetic using a redundant Montgomery representation," IEEE Transactions on Computers, Vol. 53, No. 11, pp. 1474-1482, Nov, 2004.

[9] E Savas, and CK Koc, " The Montgomery modular inverse – revisited," IEEE Transactions on Computers, Vol. 49, No. 7, pp. 763-766, Jul, 2000.

[10] Sung-Ming Yen, "Improved common-multiplicand multiplication and fast exponentiation by exponent decomposition," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. 80, No. 6, pp. 1160-1163, Jun, 1997.

[11] Duo-Li Zhang, Ming-Lun Gao, Li Li, Zuo-Ren Cheng, and Xiao-Lei Wang, "An implementation method of a RSA crypto processor based on modified Montgomery algorithm," Proceedings. 5th International Conference on ASIC, Vol. 2, pp. 1332-1336, Oct, 2003.

[12] Zhu Yao, Cheng-Hua Yan, Yu-Wei Chen, and Wei Liao, "Efficient acceleration of RSA algorithm on GPU," , 2012 IEEE International Conference on Oxide Materials for Electronic Engineering (OMEE), pp. 531-534, Sep, 2012.

[13] Sung-Ming Yen, Seung-joo Kim, and Seon-gan Lim, Sang-Jae Moon, "RSA speedup with Chinese remainder theorem immune against hardware fault cryptanalysis," IEEE Transactions on Computers,, Vol. 52, No 4, pp. 461-472, Apr, 2003.

[14] Werner Schindler, "A timing attack against RSA with the Chinese remainder theorem," Cryptographic Hardware and Embedded Systems Conference—CHES 2000, Vol. 1965, pp. 109-124, Aug, 2000.

[15] Grossschadl Johann, "The Chinese remainder theorem and its application in a high-speed RSA crypto chip," 16th Annual Computer Security Applications Conference, pp. 384-393, Dec, 2000.

[16] David Pearson, "A parallel implementation of RSA," The Conference on Selected Areas in Cryptography (SAC96), pp. 107-116, Jul, 1996.

[17] Xue-Wen Tan, and Yun-Fei Li, "Parallel analysis of an improved RSA algorithm," 2012 International Conference on Computer Science and Electronics Engineering (ICCSEE), Vol. 1, pp. 318-320, Mar, 2012.

[18] Chung-Hsien Wu, Jin-Hua Hong, and Cheng-Wen Wu, " VLSI design of RSA cryptosystem based on the Chinese remainder theorem," Journal of Information Science and Engineering, pp. 967-980, Jul, 2001.

[19] Michael J Flynn, Computer Architecture, John Wiley & Sons, Inc. 1995.

[20] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," Queue - GPU Computing, Vol. 6, No. 2, pp. 40–53, Mar, 2008.

[21] NVIDIA Corporation. NVIDIA Compute Unified Device Architecture Programming Guide Version 2, 2009.

[22] Guang Zhao, and Hen-Bo Li, "An efficient variant of the batch RSA cryptosystem," Proceedings of the 2012 International Conference on Communication, Electronics and Automation Engineering , Vol. 181, pp. 947-954, 2011.

[23] Debra L. Cook, John Ioannidis, Angelos D. Keromytis and Jake Luck, " CryptoGraphics: secret key cryptography using graphics cards," The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005. Proceedings, Vol. 3376, pp. 334-350, 2002.

[24] J-J. Quisquater, and C. Couvreur, "Fast decipherment algorithm for RSA public-key cryptosystem," Electronics Letters, Vol 18, No. 21, pp. 905-907, Oct, 1982.

[25] L. C. K. Hui and K.Y. Lam, "Fast square-and-multiply exponentiation for RSA," Electronics Letters, Vol. 30, No. 7, pp. 1396-1397, Aug, 1994.

[26] Chris J. Mitchell, Another improvement to square-and-multiply exponentiation, Royal Holloway University of London, Department of Computer Science, 1993.

[27] K. Fatahalian, J. Sugerman, and P. Hanrahan, "Understanding the efficiency of GPU algorithms for matrix multiplication," HWWS

'04 Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, pp. 133-137, 2004.

[28] Yun-Fei Li, Qing Liu, and Tong Li, "Two efficient methods to speed up the batch RSA decryption," Advanced Computational Intelligence (IWACI), 2010 Third International Workshop, pp. 469-473, Aug, 2010.

[29] Qing Liu, Yun-Fei Li, Tong Li, and Lin Hao, "The research of the batch RSA decryption performance," Journal of Computational Information Systems, pp. 948-955, Mar, 2011.

[30] Timothy G. Mattson, Beverly A. Sanders and Berna L. Massingill, Patterns for Parallel Programming, M. Addison-Wesley Professional, Sep, 2004.